SLAM
Introduction & Data Structures

EDAA - G06

Henrique Ribeiro
João Costa – João Martins
Tiago Duarte

Problem Definition

What's already done

- Filter noise/undesirable effects in data:
  - Multiple reflections;
  - Echos;
  - Self reflections;
  - Multipath errors.
- Represent map using Octomaps;
- Implement dynamic probabilistic mapping algorithms based on sonar data;

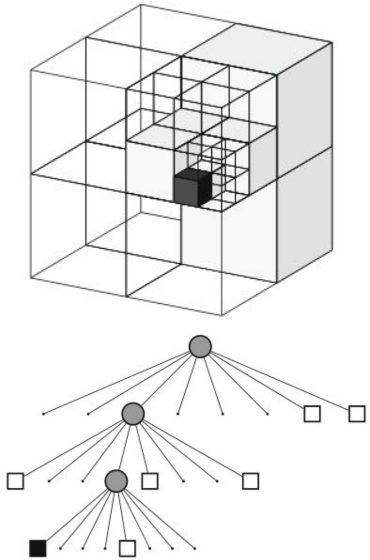

Fig 1. Example of octree storing occupied (black) and free (white) cells (Hornung et al.)

What’s next in plan

- Expand the implementation to support 3D mapping;
- Support incremental map growth;
- Improve performance;
- Implement new data structures;
    - Hash Sets/Hash Maps
- Implement new algorithms;
    - Set collision resolution techniques;
    - Explore ray casting solutions for 3D;
    - Explore more edge detection algorithms;

Data structure

Sets using Hash Maps

- The operation for set merge uses about 60% of ray cast’s execution time;
- Simple to implement;
- Low space usage;
- Can be tinkered with to better fit our problem definition.

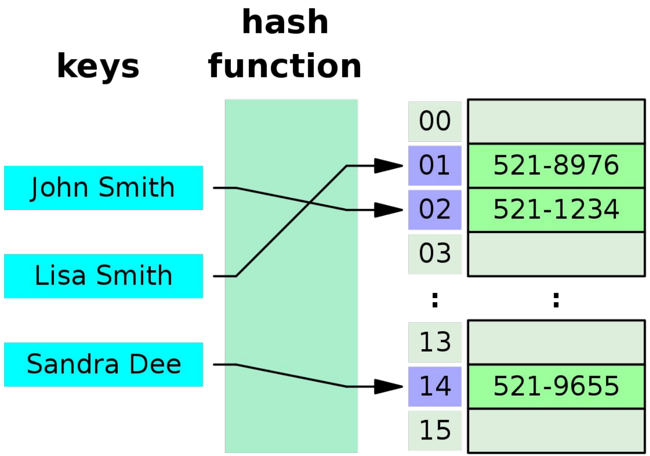

Fig 2. Hash Map Representation

Hash Sets Details

- The index to store the value corresponds to its key and is calculated using the following formula: hash % size.
- Each Hash Map only needs to maintain a list of its stored elements and the number of elements it's currently storing.
- Stores each element once and only once.
- The elements to be stored need a good hash function;
- Set collision resolutions technics change the way elements with the same hash are stored.

Key 1	Value 1
Key 2	Value 2
Key 3	Value 3

Fig 3. Hash Map Reference

Collision Resolution - Linear

- If two elements have the same hash, the next index is tested (essentially **hash = hash + 1**);
- Very simple to implement and verify;
- Not the most efficient algorithm.
- Poor hash functions affect this algorithm the most.
- **Deleted node is not removed, but marked as “removed”.**

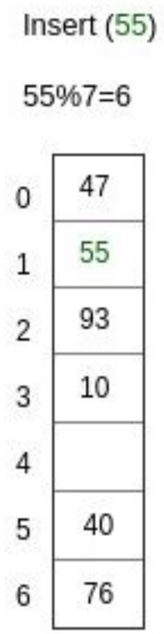

Fig 4. Linear Probing

Collision Resolution - Quadratic

- If two elements have the same hash, we try **index = hash + nColission^2**.
- Also easy to implement, but can be a bit harder to verify;
- Can create **fewer** collisions than linear probing.
- **Deleted node is not removed, but marked as “removed”.**

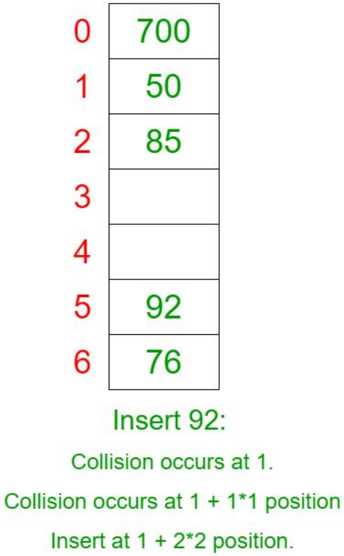

Fig 5. Quadratic Probing

Collision Resolution - Double Hashing

- If two elements have the same hash, we try **hash + nColission*hash**.
- Hard to verify;
- Creates **fewer** collisions in hashes with the same index. The next index to test in these cases will be different, contrary to the previous algorithms.
- **Deleted node is not removed, but marked as “removed”.**
- Hash must **never** equal 0.

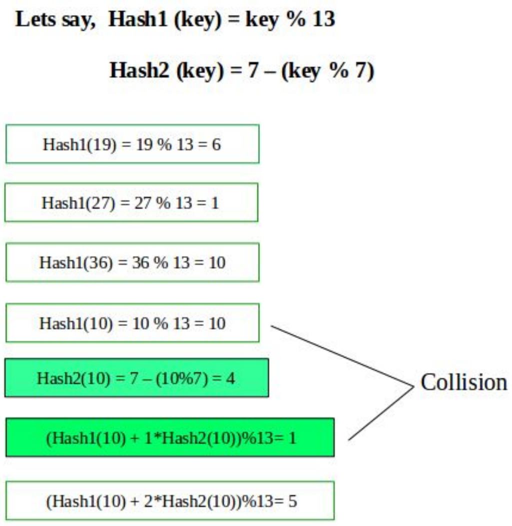

Fig 6. Quadratic Probing

Collision Resolution - Separate Chaining

- If two elements have the same hash, we store both in a **bucket**.
- **Harder** to implement but **easy** to verify;
- Collisions are handled **easily** and thus, the insert operation works the **fastest** in this algorithm.
- **Deleted node can be deleted.**
- Uses **more** space as there are additional data structures.
- **Open hashing**, which means cashing is harder, thus leading to more cache misses.

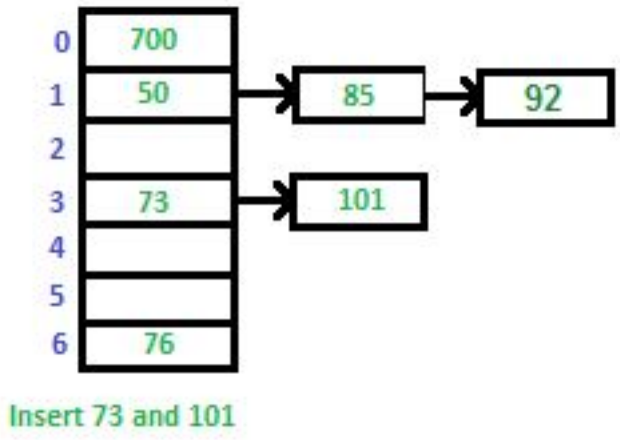

Fig7. Separate Chaining

HashMap Resizing

- When a certain **threshold** of occupied indexes is reached, the map needs to be **resized**.
- This leads to **fewer** hash collisions since there are more possible indexes to store the values.

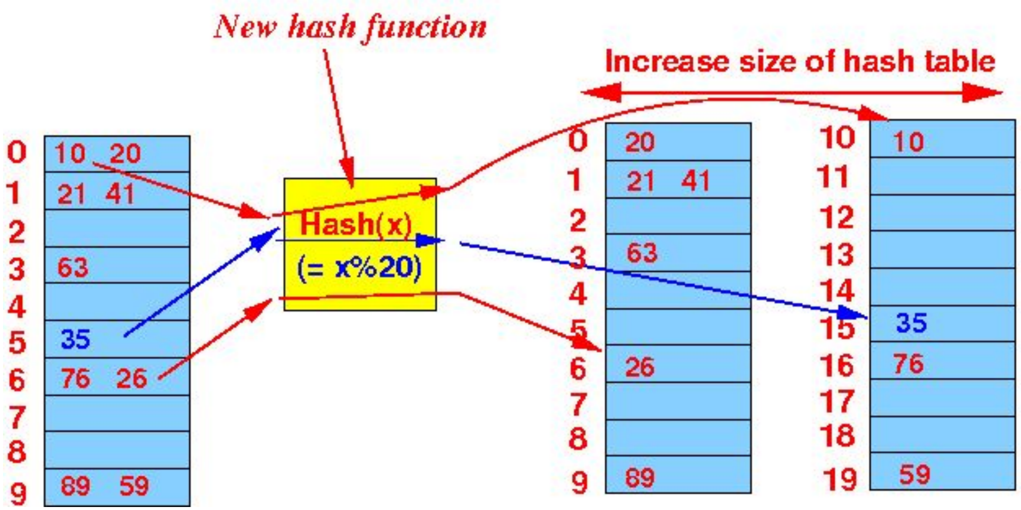

Fig 8. Resizing HashMap

HashMap Resizing

- After **resize**, the previously stored values need to be relocated.
- This can lead to a **high overhead** since we need to calculate the hash of the values all over again.
- To mitigate this problem, **the hash of each element is stored** alongside it, so it can be reutilized when calculating the new index position

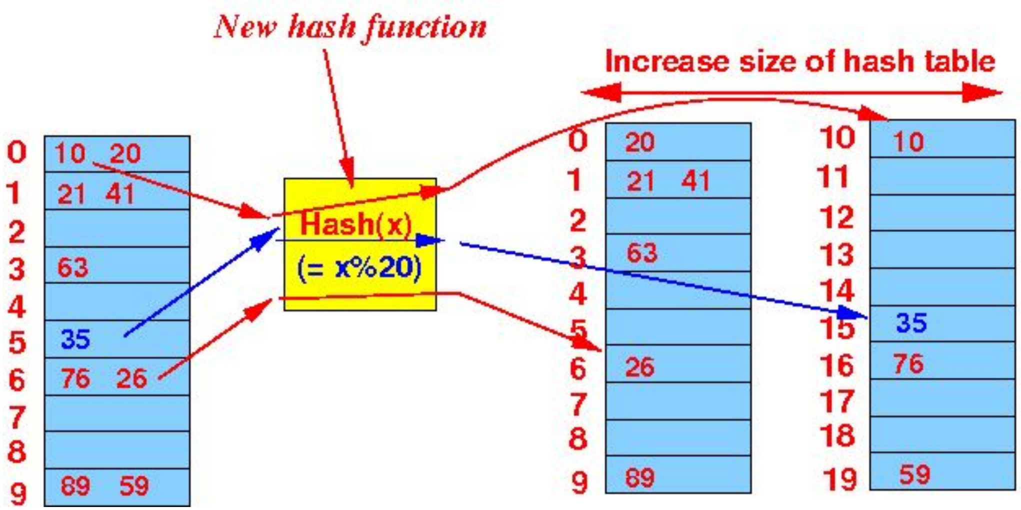

Fig 9. Resizing HashMap

HashMaps in Our Project

- Our **Hashmaps** will be used instead of STD hashmaps:
  - The STD hashmaps were made to be as generic as possible, and this can lead to some overhead in our project (the merge of sets take about 60% of raycast’s execution time).
- The initial size can be **calculated** to reduce the number of resizings done:
  - By using the number of raycasts we want to perform, we can estimate the number of elements each hashmap will have.
- It is **uncertain** which collision resolution technique will prove to be the best
  - They will be measured and compared.

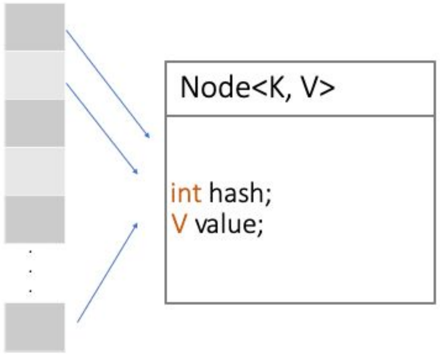

Fig 10. Example of a table entry in our implementation

Questions?